Computação Paralela em JavaScript

Orientador: Professor Doutor Calebe de Paula Biachini

Orientando: Clauber P. Stipkovic H.

Apoio PIVIC Mackenzie

1. ABSTRACT

The objective of this scientific research have its starting point to present how a JavaScript engine works and more specifically the Mozilla's JavaScript engine called SpiderMonkey, investigate and if possible apply parallel computing concepts and techniques this JavaScript engine.

2. INTRODUÇAO

Nos últimos 20 anos, a *World Wide Web* (conhecida popularmente como Web) se tornou uma ferramenta absolutamente importante para o mundo moderno de modo geral, mudando o modo como as pessoas se comunicam, fazem negócios, estudam entre outras atividades cotidianas, provendo produtos e serviços para suprir essas necessidades, encurtando distancias e facilitando a vida dos usuários.

Desde a chamada “guerra dos navegadores”, em meados de 1995, entre os *web browsers* **Microsoft Internet Explorer** e **Netscape Navigator**, quando houve a disputa por mercado entre os dois web browser, até hoje, o foco dos desenvolvedores de aplicações mudou consideravelmente para acompanhar a transição de aplicações tipicamente desktop, para aplicações *multi-device*.

No começo dos anos 90, muitas aplicações era desenvolvidas apenas e exclusivamente para funcionarem nos *desktops*; porém, com o passar dos anos, a chegada dos *web browsers* e, posteriormente, a evolução das linguagem web e dos *devices portáteis* fez com que os desenvolvedores e as empresa ligadas a tecnologia, passassem a ver a web como uma plataforma com um imenso potencial

consequentemente, para criação de novas aplicações e para a disponibilização de aplicações já existentes nos desktops serem levadas para os web browsers.

Com a popularização da internet (sistema global de redes de computadores que utilizam o padrão de protocolos TCP/IP), a facilidade para compra de computadores e ao acesso a web, fez com que muitos desenvolvedores e empresas surgissem focadas nas aplicações para a web.

Por volta do ano de 2005, com o surgimento do conceito de Web 2.0, que tinha como ponto de foco a web como plataforma, mas também mostrava por que muitas das empresas sobreviveram a bolha especulativa da internet no final da década de 90, muitos produtos e serviços como leitores de e-mail, aplicações de agencias bancarias, etc, começaram a nascer voltados para a web, e não mais exclusivamente para desktops, como antes.

Devido a grande utilização da web como plataforma e sua popularização, a demanda por web browsers mais rápidos e eficientes fez com que as tecnologias que dão suporte para essas aplicações fossem se tornando mais importantes a cada dia.

Com isso, veio também o uso massivo da linguagem JavaScript, que foi criada como forma de permitir mais dinamismo para as web pages e também interatividade entre os web browsers (chamados de client-side) e servidores de dados para as aplicações (chamados de server-side).

Por verificar a importância de termos uma JavaScript engine que seja rápida e eficiente para gerenciar o uso da CPU dos devices onde e utilizada, esta pesquisa utilizou como base a JavaScript engine da Fundação Mozilla, chamada SpiderMonkey, onde foram realizados os estudos para entender como se da o funcionamento de uma JavaScript engine e com isso, identificar possíveis pontos onde fosse possível aplicar conceitos de computação paralela.

3. A LINGUAGEM JAVASCRIPT

A linguagem JavaScript foi criada em 1995 pelo desenvolvedor Americano, Brendan Eich, como parte do web browser Netscape Navigator para que este, tivesse a

habilidade de interpretar scripts client-side.

É uma linguagem de programação dinâmica e multiparadigma (script, orientada a objetos, imperativa e funcional), conhecida por ser utilizada para interagir com o lado cliente dos web browser (através da plataforma Document Object Model), comunicação assíncrona e mais atualmente por também haver a possibilidade de ser executada do lado servidor, através do projeto NodeJS.

Seu desenvolvimento teve como base a padronização da especificação ECMA-2623 e ISO/IEC 16262. Ao final de sua implementação a Netscape (atualmente Mozilla) submeteu em novembro de 1996 a especificação da linguagem JavaScript, que foi aceita e teve a definição do padrão chamada de ECMAScript.

A partir de 2012 os browsers mais modernos como Mozilla Firefox, Google Chrome e Opera Browser passaram a dar suporte completo ao ECMAScript 5.1. Em Junho 17 de Junho de 2015, a ECMAScript 6 foi publicada oficialmente e é popularmente chamada pelos desenvolvedores de ES6.

4. DEFINIÇÃO DE UMA JAVASCRIPT ENGINE

Uma JavaScript engine, também citada como JavaScript interpreter ou JavaScript implementation, é uma *process virtual machine* que interpreta e executa códigos JavaScript.

> “A process Virtual Machine, sometimes called an *application virtual machine*, or *Managed Runtime Environment* (MRE), runs as a normal application inside a host OS and supports a single process. It is created when that process is started and destroyed when it exits. Its purpose is to provide a platform-independent programming environment that abstracts away details of the underlying hardware or operating system, and allows a program to execute in the same way on any platform.” - https://en.wikipedia.org/wiki/Virtual_machine#Process_virtual_machines

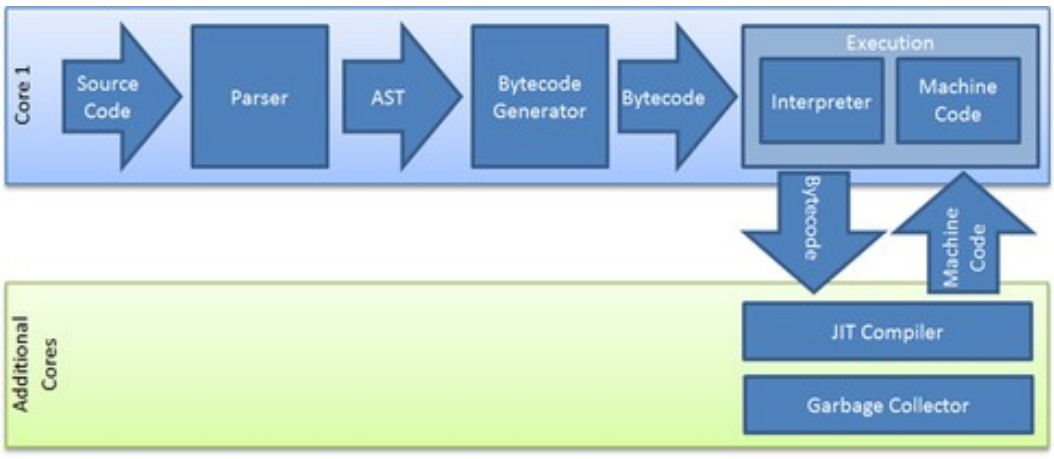
Por definição, sua base de construção segue o mesmo princípio de uma *process virtual machine*, e que contem passos que são comuns e definidos para todo tipo de virtual machine:

- Parser
- Intermediate Representation (IR)
- Interpreter
- Garbage Collection
- Optimization

Originalmente, a primeira JavaScript Engine foi construida para possibilitar a execucao de codigo JavaScript no *web browser* Netscape Navigator, mas atualmente, e utilizada tambem para execucao de codigos JavaScript em CLI (Command Line Interface), o que possibilita utilizar a linguagem tambem em *clint-side*

4.1 FLUXO DE FUNCIONAMENTO DA JAVASCRIPT ENGINE

O esquema teórico abaixo, mostra o fluxo que um código JavaScript percorre na JavaScript engine, rodando em um processador multi-core, desde recebimento do código até a sua execução.



Partindo do inicio no caminho percorrido pelo código JavaScript, como mostrado no esquema acima, utilizamos um trecho de código que faz uma soma entre dois números e retorna o resultado dessa soma, como prova de conceito para mostrar passo a passo os resultados de cada etapa do processo na JavaScript engine.

```
1 function sum(a, b) {
2     return (a + b);
3 }
4
```

Ao receber o trecho de código, a JavaScript engine inicia o parsing (conhecido em português como analise sintática), processo que consiste em analisar uma sequencia de símbolos ou caracteres em linguagem natural ou de computador, construindo uma estrutura hierárquica correspondente ao que foi informado antes do método de parsing ser aplicado.

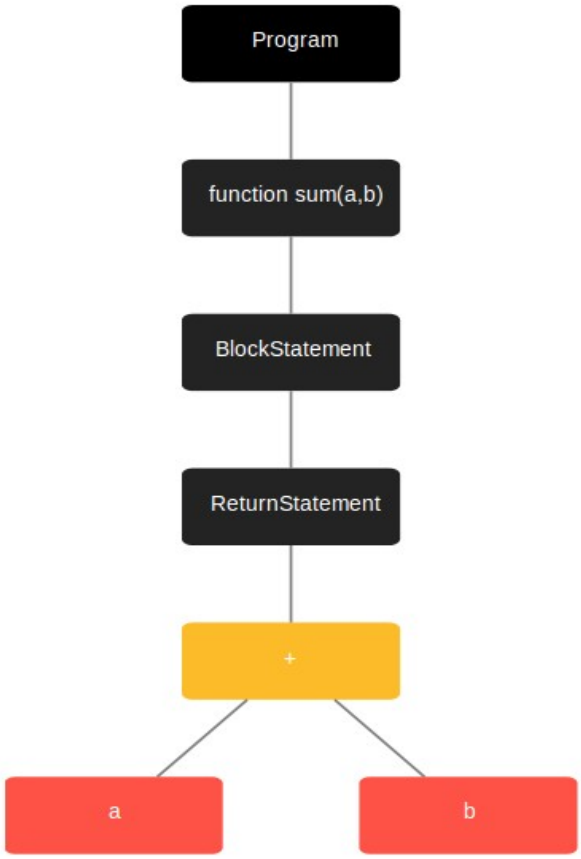
É durante o processo de parsing também que a validação sintática do código, procurando por possíveis erros baseado na escrita padrão da linguagem é feito, para evitar erros durante a compilação ou resultados divergentes na execução do código ao final do processo.

Utilizado a função de soma como exemplo, o resultado do parsing utilizando o método *tokenizer*[1], temos o resultado do trecho de código da função soma que será gerado é:

```
# of nodes: 7
# of tokens: 16
Tokens:
        Keyword(function)  Identifier(sum)
        Punctuator(()  Identifier(a)  Punctuator(,)
        Identifier(b)  Punctuator())  Punctuator({)
        Keyword(return)  Punctuator(()
        Identifier(a)  Punctuator(+)  Identifier(b)
        Punctuator())  Punctuator(;)
        Punctuator(})
```

Após a geração Com isso, uma abtract syntax tree (AST) é formada, como no exemplo para a função de soma:

1 Tokenizer pode ser um programa que executa analizes lexicas em uma seguencia de caracteres. (Jinks,Pete - http://www.cs.man.ac.uk/~pjj/farrell/comp3.html, Anatomy of a Compiler)

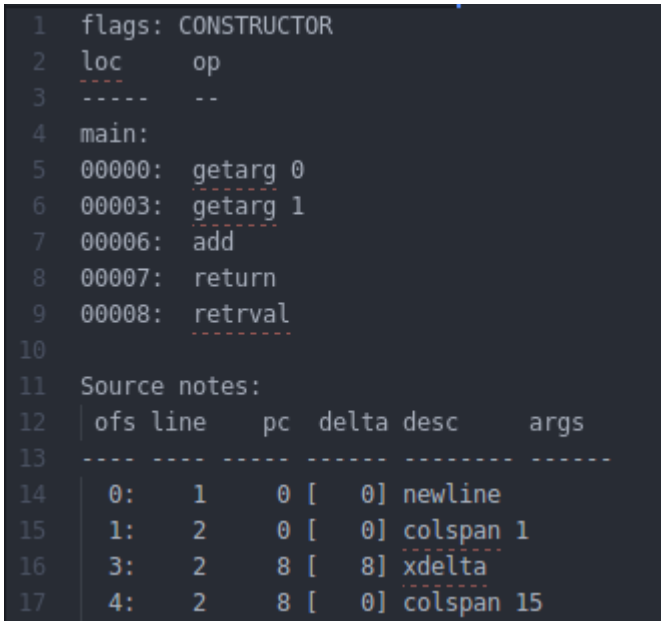


Uma vez que a abstract syntax tree (AST) é formada, com base no tipo da JavaScript engine, o bytecode generator converte a AST para uma linguagem intermediaria ou código nativo[2], e isso é feito para cada bloco de código dentro da AST.

Bytecode, também conhecidos como **portable code**, são formar canónicas de representação de códigos (também chamados de Intermediate Representation) que são projetados para obter execução eficiente por software interpretador.

Seguindo o passo de interpretação e compilação, abaixo podemos verificamos o bytecode gerado para a função “sum” que utilizamos como referencia:

2 A etapa de geracao do bytecode, pode variar nas diferentes implementacoes de JavaScript engine, como por exemplo a Rhino (https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Rhino/JavaScript_Compiler), que por ser escrita em lingaugem Java, adiciona a etapa de traducao do codigo JavaScript para classes Java.



```
flags: CONSTRUCTOR
loc     op
-----   --
main:
00000:  getarg 0
00003:  getarg 1
00006:  add
00007:  return
00008:  retrval

Source notes:
 ofs line    pc  delta desc     args
---- ---- ----- ------ -------- ------
  0:    1     0 [   0] newline
  1:    2     0 [   0] colspan 1
  3:    2     8 [   8] xdelta
  4:    2     8 [   0] colspan 15
```

Após a geração do bytecode, entramos na fase de interpretação e execução da representação intermediária, o que é comumente feito de utilizando uma função simples, em vários passos, uma instrução por vez percorrendo o bytecode gerado.

Quando o bytecode chega até a area de execução, observamos a existência de dois componentes importantes para a performance de uma JS engine, que são o JIT (Just-in-Time) Compiler e o Garbage Collector.

O JIT, é responsável por traduzir bytecodes para código de maquina durante a execução do programa, ou seja, assim que um trecho de código é requisitado, o JIT transforma o bytecode para código de maquina, referente ao bloco solicitado e o disponibiliza para execução, fazendo com que a disponibilização do resultado seja rápida e não consuma ciclos de processamento desnecessários entregando somente os blocos que serão utilizados no momento em que forem requeridos.

Junto ao JIT, encontramos o Garbage Collector (ou coletor de lixo), que é a área responsável pelo gerenciamento automático da memória que é ocupada por objetos, sendo denominados como “lixo” as posições de memória que estão sendo ocupadas com informações referentes ao programa em execução mas que não são mais relevantes ou que não são mais utilizadas.

Tanto o JIT quando o Garbage Collector, são executados quando solicitados, e estão disponíveis durante toda a execução da JS engine, por isso estão diretamente

ligados ao passo de execução dentro do esquema apresentado.

5. A JAVASCRIPT ENGINE SPIDERMONKEY[3]

Escrita utilizando linguagens de programação como C, C++ e JavaScript, pode ser executado em vários sistemas operacionais e devices como celulares, tablets e até mesmo em aparelho de TV, o que a torna muito adaptável e com robustez.

5.1 ESTRUTURA DA JAVASCRIPT ENGINE SPIDERMONKEY

Por ser uma JavaScript engine, a SpiderMonkey possui sua estrutura como uma *process virtual machine* e segue um fluxo de interpretação de códigos JavaScript muito próxima ao apresentado no tópico ***Fluxo de funcionamento da JavaScript engine.***

Mesmo tendo usa base como sendo uma JavaScript engine, sua estrutura contem algumas implementações que visam melhorar o desempenho e performance na geração e interpretação de códigos JavaScript, bem como otimizar o gerenciamento e de uso de memória durante sua execução.

5.2 JUST-IN-TIME COMPILER NA SPIDERMONKEY

Como uma forma de melhorar o desempenho durante a execução do JavaScript, existem duas camadas de JITs que funcionam separadamente e com objetivos diferentes.

A primeira camada, conhecida apenas como Baseline Compiler, e que tem como função apenas uma compilação preliminar, sem otimizações durante a execução, e foi introduzida para substituir a antiga camada method JIT chamada JaegerMonkey, com o objetivo de fácil manutenção e por possibilitar a otimização de novas funcionalidades na linguagem JavaScript.

---

3 A JavaScript engine SpiderMonkey é um projeto de código aberto criado por Brendan Eich, na Netscape Communications em meados de 1996, e é considerada a primeira JavaScript engine da história.

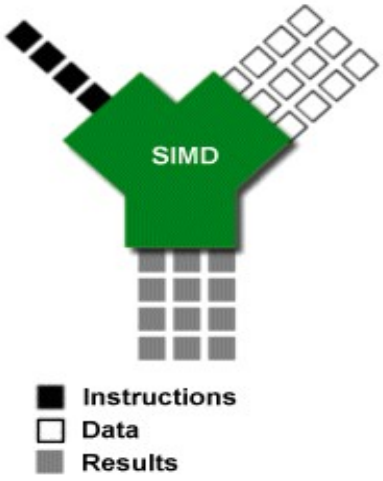
A segunda camada, chamada de IonMonkey, foi desenvolvido com foco em performance e otimização na geração do código, sendo inteiramente um method JIT.

A JIT Compiler IonMonkey

6. SINGLE INSTRUCTION, MULTIPLE DATA (SIMD)

“Single Instruction, multiple data”, ou comumente chamado de SIMD, faz parte de uma das quatro classes da que são definidas pela taxonomia de Flynn na arquitetura de computadores, para descrever computadores com processamento múltiplo, como parte da computação paralela[4].

SIMD pode ser classificado como um caso especial de um programa com um fluxo único de entrada de instruções, múltiplas fluxos de entrada de dados e múltiplas saídas de resultados, o que é chamado de paralelismo de dados e que é mostrado na imagem abaixo.



7. A API SIMD.JS

A interface de programação de aplicações SIMD.js esta sendo desenvolvida por

4 “Parallel computing is the use of two or more processors (cores, computers) in combination to solve a single problem” (Stout, Quentin F. http://web.eecs.umich.edu/~qstout/parallel.html, acessado em 10/05/2015)

empresas como Intel, Google e Mozilla com a intenção de inserir novos tipos e funções paralelas na linguagem JavaScript.

Uma das inclusões é referente ao tipo Float32x4, que consiste em representar 4 valores float32, conhecido como Single-precision floating-point format, que podem ser enviados simultaneamente para a JavaScript Engine, pois a SIMD.js possui todas as operações aritméticas básica e operações para rearranjar, carregar e armazenar esses valores.

Atualmente o foco do desenvolvimento é suportar tanto arquiteturas x86 com Stream SIMD Extensions[5] quanto arquiteturas ARM com tecnologia NEON[6].

8. CONCLUSAO

Ao término deste projeto de iniciação científica, percebemos que existem muitos esforços para que cada vez mais recursos de computação paralela sejam utilização na linguagem JavaScript.

No momento da publicação desse projeto, estamos caminhando para

- https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey/Hacking_Tips

- https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey/Parser_API

- http://www.slideserve.com/oriana/v8-an-open-source-high-performance-javascript-engine

- http://www.slideshare.net/nwind/virtual-machine-and-javascript-engine?qid=683d6d16-5fd1-4157-88ef-4936dac92217&v=qf1&b=&from_search=2

- http://www.slideshare.net/shadedecho/javascript-architecture-the-front-and-the-back-of-it-3518425

- http://www.slideshare.net/axemclion/understanding-javascript-engines

- http://www.slideshare.net/RednaxelaFX/implement-js-krystalmok20131110

- http://www.slideshare.net/nwind/javascript-engine-performance?qid=683d6d16-5fd1-4157-88ef-4936dac92217&v=qf1&b=&from_search=4

- http://www.slideshare.net/RednaxelaFX/implement-js-krystalmok20131110?qid=683d6d16-

5 https://en.wikibooks.org/wiki/X86_Assembly/SSE
6 http://www.arm.com/products/processors/technologies/neon.php

5fd1-4157-88ef-4936dac92217&v=qf1&b=&from_search=6

- http://www.slideshare.net/lijing00333/javascript-engine?qid=683d6d16-5fd1-4157-88ef-4936dac92217&v=qf1&b=&from_search=7

- http://www.slideshare.net/amdgigabyte/know-your-javascript-engine?qid=683d6d16-5fd1-4157-88ef-4936dac92217&v=qf1&b=&from_search=8

- http://www.slideshare.net/senchainc/javascript-engines-under-the-hood?qid=683d6d16-5fd1-4157-88ef-4936dac92217&v=qf1&b=&from_search=12

- https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Rhino/JavaScript_Compiler

- https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey/JSAPI_reference

- https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey/JSAPI_User_Guide

- https://wiki.mozilla.org/JavaScript:New_to_SpiderMonkey

- https://en.wikipedia.org/wiki/Just-in-time_compilation

- https://en.wikipedia.org/wiki/JavaScript_engine

- http://wiki.jvmlangsummit.com/images/c/ce/Nashorn.pdf

https://en.wikipedia.org/wiki/List_of_ECMAScript_engineshttps://blog.mozilla.org/javascript/2013/04/05/the-baseline-compiler-has-landed/

- https://github.com/Benvie/continuum#continuum---a-javascript-virtual-machine-built-in-javascript

- https://en.wikipedia.org/wiki/Programming_paradigm#Multi-paradigm

- http://tratt.net/laurie/research/pubs/html/tratt__dynamically_typed_languages/

- https://en.wikipedia.org/wiki/SpiderMonkey_%28software%29

- https://blog.mozilla.org/javascript/page/2/

- https://blog.mozilla.org/javascript/2014/01/23/the-monkeys-in-2013/

-

- https://blog.mozilla.org/javascript/2012/08/22/hello-world/

- https://blog.mozilla.org/javascript/2015/02/26/the-path-to-parallel-javascript/

- http://www.umiacs.umd.edu/research/parallel/

9. BIBLIOGRAPHY

1. Internet (https://en.wikipedia.org/wiki/Internet)
2. Internet Protocol Suite – TCP/IP (https://en.wikipedia.org/wiki/Internet_protocol_suite)
3. Introduction to the Internet Architecture – RFC1122, October 1989 (https://tools.ietf.org/html/rfc1122#page-7)
4. Browser War I (http://browserwars.wikia.com/wiki/Browser_War_I)
5. What Is Web 2.0 - Design Patterns and Business Models for the Next Generation of Software by Tim O'Reilly – 30/09/2005 (http://www.oreilly.com/pub/a/web2/archive/what-is-web-20.html)
6. James K. Galbraith and Travis Hale (2004). Income Distribution and the Information Technology Bubble. University of Texas Inequality Project Working Paper (http://utip.gov.utexas.edu/papers/utip_27.pdf)
7. ECMAScript® 2015 Language Specification - 6thEdition / June 2015 (http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf)
8. Mozilla Developer Network – JavaScript (https://developer.mozilla.org/en-US/docs/Web/JavaScript)
9. JavaScript (https://en.wikipedia.org/wiki/JavaScript)
10. SpiderMonkey (https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey)
11. How does a JavaScript engine work? (https://www.quora.com/How-does-a-JavaScript-engine-work/answer/Amar-Prabhu)
12. Lexical analysis (https://en.wikipedia.org/wiki/Lexical_analysis)
13. SpiderMonkey (software) (https://en.wikipedia.org/wiki/SpiderMonkey_%28software%29)
14. SpiderMonkey Internals (https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey/Internals)
15. Just-in-time compilation (https://en.wikipedia.org/wiki/Just-in-time_compilation)
16. JavaScript Engine (http://en.wikipedia.org/wiki/JavaScript_engine)
17. Dynamically typed languages, Laurence Tratt, Advances in Computers, vol. 77, pages 149-184, July 2009

(http://tratt.net/laurie/research/pubs/html/tratt__dynamically_typed_languages/)

18. Dynamic programming language (https://en.wikipedia.org/wiki/Dynamic_programming_language)

19. A Short History of JavaScript (https://www.w3.org/community/webed/wiki/A_Short_History_of_JavaScript)

20. JavaScript – The Definite Guide, Sixth Edition, David Flanagan, March 2011, O'Reilly

21. Compilers: Principles, Technique, and Tools, Second Edition, Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, Addison Wesley, 2007

22. JSAPI User Guide (https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey/JSAPI_User_Guide)https://en.wikipedia.org/wiki/Parsing

23. Work With JS Engine API (http://www.scribd.com/doc/269666132/Work-With-JS-Engine-API)

24. Introduction to the JavaScript Shell (https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey/Introduction_to_the_JavaScript_shell)

25. SpiderMonkey Build Documentation (https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey/Build_Documentation)

26. Parsing ()

27. Abstract Syntax Tree (https://en.wikipedia.org/wiki/Abstract_syntax_tree)

28. JavaScript AST visualizer (http://jointjs.com/demos/javascript-ast)

29. What is JavaScript AST, how to play with it? (http://stackoverflow.com/questions/16127985/what-is-javascript-ast-how-to-play-with-it)

30. How browsers work (http://taligarsiel.com/Projects/howbrowserswork1.htm)

31. Bytecode (https://en.wikipedia.org/wiki/Bytecode)

32. Bytecodes (https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey/Internals/Bytecodes)

33. A LISP garbage-collector for virtual-memory computer system (http://dl.acm.org/citation.cfm?id=363280)

34. Garbage collection (computer science) (https://en.wikipedia.org/wiki/Garbage_collection_%28computer_science%29)

35. A parallel, real-time garbage collector (http://dl.acm.org/citation.cfm?id=378823)

36. JSWhiz - Static Analysis for JavaScript Memory Leaks (http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/40738.pdf)

37. The Baseline Compiler Has Landed (https://blog.mozilla.org/javascript/2013/04/05/the-baseline-compiler-has-landed/)

38. IonMonkey – Mozilla Wiki (https://wiki.mozilla.org/IonMonkey)

39. IonMonkey: Mozilla's new JavaScript JIT compiler (http://www.infoq.com/news/2011/05/ionmonkey)

40. Platform/Features/IonMonkey (https://wiki.mozilla.org/Platform/Features/IonMonkey)

41. Flynn, M. J. (September 1972). "Some Computer Organizations and Their Effectiveness". IEEE Trans. Comput. C–21 (9): 948–960. doi:10.1109/TC.1972.5009071

42. SIMD (https://en.wikipedia.org/wiki/SIMD)

43. What do Sony's Playstation2 and Motorola's MPC7400 (http://arstechnica.com/features/2000/03/simd/)

44. IntelLabs/RiverTrail (https://github.com/IntelLabs/RiverTrail)

45. IntelLabs/RiverTrail Wiki (https://github.com/IntelLabs/RiverTrail/wiki)

46. Parallel JavaScript (https://software.intel.com/en-us/blogs/2011/09/15/parallel-javascript)

47. RiverTrail – JavaScript Engine (https://en.wikipedia.org/wiki/River_Trail_%28JavaScript_engine%29)

10. REFERENCIAS

11. - Mozilla Source Code Directory Structure (https://developer.mozilla.org/en-US/docs/Mozilla_Source_Code_Directory_Structure)

- What is the Document Object Model? (http://www.w3.org/DOM/#what)

- NodeJS (https://nodejs.org/)

12. ANEXO

- Mozilla Public License Version 2.0 (https://www.mozilla.org/MPL/2.0/)

- Duck Typing (https://en.wikipedia.org/wiki/Duck_typing)